

PROGRAMMABLE LOGIC DEVICE WITH
METHOD OF PREVENTING READBACK

Raymond C. Pang
Walter N. Sze
John M. Thendean
Stephen M. Trimberger
Jennifer Wong

FIELD OF THE INVENTION

The invention relates to PLDs, more particularly to protection of designs loaded into a PLD through a bitstream.

BACKGROUND OF THE INVENTION

A PLD (programmable logic device) is an integrated circuit structure that performs digital logic functions selected by a designer. PLDs include logic blocks and interconnect lines and typically both the logic blocks and interconnections are programmable. One common type of PLD is an FPGA (field programmable logic device), in which the logic blocks typically include lookup tables and flip flops, and can typically generate and store any function of their input signals. Another type is the CPLD (complex programmable logic device) in which the logic blocks perform the AND function and the OR function and the selection of input signals is programmable.

Problem with storing bitstream external to PLD

Designs implemented in PLDs have become complex, and it often takes months to complete and debug a design to be implemented in a PLD. When the design is going into a system of which the PLD is a part and is to be sold for profit, the designer does not want the result of this design effort to be copied by someone else. The designer often wants to keep the design a trade secret. Many PLDs, particularly FPGAs, use volatile configuration memory that must be loaded from an external device such as a PROM every time the PLD is powered up. Since configuration data is stored external to the PLD and must be transmitted through

09724975-12800

EL539650805US

4

Efforts have been made to encrypt designs, but it is difficult to make the design both secure from attackers and easy to use by legitimate users. The encryption algorithm is not a problem. Several encryption algorithms, for example, the standard Data Encryption Standard (DES) and the more secure Advanced Encryption Standard (AES) algorithm, are known for encrypting blocks of data. The process of cipher block chaining (CBC), in which an unencrypted data word is XORed with the next encrypted data word before decryption allows the DES or AES to encrypt a serial stream of data and these are therefore appropriate for encrypting a bitstream for configuring a PLD. A key used for encrypting the design must somehow be communicated in a secure way between the PLD and the structure that decrypts the design, so the design can be decrypted by the PLD before being used to configure the PLD. Then, once the PLD has been configured using the unencrypted design, the design must continue to be protected from unauthorized discovery.

20 A November 24, 1997 publication by Peter Alfke of Xilinx, Inc. entitled
21 “Configuration Issues: Power-up, Volatility, Security, Battery Back-up” describes
22 several steps that can be taken to protect a design in an existing FPGA device
23 having no particular architectural features within the FPGA to protect the design.
24 Loading design configuration data into the FPGA and then removing the source of
25 the configuration data but using a battery to maintain continuous power to the
26 FPGA while holding the FPGA in a standby non-operational mode is one method.
27 However, power requirements on the battery make this method impractical for
28 large FPGA devices.

Nonvolatile configuration memory is another possibility. If the design is loaded at the factory before the device is sold, it is difficult for a purchaser of the

000001137

1 configured PLD device to determine what the design is. However, a reverse
2 engineering process in which the programmed device is decapped, metal layers
3 are removed, and the nonvolatile memory cells are chemically treated can expose
4 which memory cells have been charged and thus can allow an attacker to learn the
5 design. Further, nonvolatile memory requires a more complex and more
6 expensive process technology than standard CMOS process technology, and takes
7 longer to bring to market.

8 It is also known to store a decryption key in nonvolatile memory in a PLD,
9 load an encrypted bitstream into the PLD and decrypt the bitstream using the key
10 within the PLD. This prevents an attacker from reading the bitstream as it is being
11 loaded into the PLD, and does retain the key when power is removed from the
12 PLD. Such an arrangement is described by Austin in U.S. Patent 5,388,157. But
13 this structure does not protect the user's design from all modes of attack.

14 In addition to design protection, some users need data protection. They may
15 have generated data within the PLD that should not be lost when the PLD loses
16 power. It is desirable to protect such data.

17 There remains a need for a design protection method that is convenient,
18 reliable, and secure.

19 20 SUMMARY OF THE INVENTION

21 The invention provides several structures and methods for protecting a PLD
22 from unauthorized use and data loss.

23 If the PLD is configured by static RAM memory that must be loaded on
24 power-up, the configuration data must be protected as it is being loaded into the
25 device. As in the prior art, this is accomplished by encrypting the configuration
26 data for storing it in a memory outside the integrated circuit device, loading one
27 or more decryption keys into the PLD and maintaining the keys in the PLD when
28 powered down, including a decryption circuit within the PLD that uses the key to
29 decrypt the configuration data, generating decrypted configuration data within
30 the PLD and configuring the PLD using the decrypted configuration data.

For additional security, rather than using nonvolatile memory to preserve keys, the invention preferably uses a battery connected to the PLD to preserve the key when power is removed from the PLD. Whereas it is possible to remove a PLD storing keys in nonvolatile memory, decap the PLD and observe which of the nonvolatile bits are programmed to logic 1 and which are programmed to logic 0, it is believed that it is very difficult to determine the contents of keys stored only in static memory cells since power must be maintained to the memory cells storing the keys in order for the keys to even be stored, and the PLD would have to be decapped, delayed, and probed while operating power is continuous to the PLD.

Ways an attacker can steal a design once loaded into a PLD

If a key does not offer sufficient security, an attacker may break the encryption code and determine the value of the key. The well-known Data Encryption Standard DES used a 56-bit encryption key, and has been broken in a few hours by a sophisticated computer to reveal the key. DES is described by Bruce Schneier in "Applied Cryptography Second Edition: protocols, algorithms, and source code in C" copyright 1996 by Bruce Schneier, published by John Wiley & Sons, Inc., at pages 265-278. If it is desirable to use such a well known encryption standard, then in order to increase security, the configuration data may be encrypted several times using different keys each time, thus strengthening the encryption code by about 2^{56} each time the encryption is repeated. Or it may be encrypted using a first key, decrypted using a second key, and encrypted using a third key, a combination that is part of the triple DES standard. Other encryption algorithms may also be used, and it is not necessary to keep the algorithm secret since the security resides in the key. When the encryption method is symmetrical, the same keys used for encryption are stored in the PLD and used in reverse order for decryption.

In a PLD offering multiple keys, if the number of keys to be used and the addresses of all keys were provided in an unencrypted bitstream, an attacker

1 might be able to attack the keys one at a time and more easily determine the key
2 values. To avoid such attack, additional security is achieved by storing within the
3 keys, not the bitstream, an indication of how many keys are to be used and
4 whether a key is the last key of a set or whether more are to follow.

5 If the PLD offers the option of reading back the bitstream after it has been
6 loaded into the PLD, another method that can be used by an attacker is to read
7 back this bitstream. To avoid this method of attacking the design, in one
8 embodiment, a PLD that offers readback and also offers encryption includes the
9 ability to disable the readback feature when encryption has been used. In another
10 embodiment, the PLD that offers the ability to read back encrypts the
11 configuration data before it is read back.

12 Additionally, some PLDs offer the option of partial configuration (where
13 several configuration addresses are specified for loading several portions of a
14 design) and partial reconfiguration (where an existing design is not erased before
15 new design data are loaded). If the PLD offers these options, an attacker could
16 partially reconfigure a PLD to make successive portions of the design visible, and
17 probably learn the whole design. To avoid such an attack, in one embodiment,
18 partial configuration and reconfiguration of PLDs loaded with encrypted designs
19 are disallowed. In another embodiment, several configuration addresses can be
20 specified, but the addresses are encrypted.

21 Yet another mode of attack is to try to flip a bit that indicates the security
22 status of the PLD. Lowering or raising the operating voltage, changing the
23 temperature, and applying noise to certain ports come to mind. To protect against
24 such bit-flipping, when the PLD is operating with a secured bitstream, a secure-
25 mode flag is set, and in one embodiment, if this flag becomes unset, all
26 configuration data is erased. In another embodiment that doesn't allow for
27 reconfiguration while the device is still operating, the configuration data is erased
28 before any bitstream is sent.

29 Another mode of attack is to relocate portions of the encrypted bitstream so
30 that when they are unencrypted they are placed into visible portions of the PLD

1 not intended by the designer. To prevent this relocation, address information is
2 used in the encryption and decryption processes so that sending a portion of an
3 encrypted bitstream to a different PLD location from that intended by the designer
4 will cause it to decrypt differently into data with no meaning. Cipher block
5 chaining (CBC) is one effective means of achieving this result. In cipher block
6 chaining, the decrypted data packet (block) is combined using the XOR function
7 with the next data block before the next block is decrypted, thus the encrypted
8 data for each data block depends on every block that preceded it and on the order
9 of those blocks. Identical blocks of data will encrypt to different values depending
10 on the value of the data blocks that preceded them. This way, if the order of the
11 blocks is changed, the bitstream will not decrypt correctly because the place where
12 the encrypted bitstream is rearranged will scramble subsequent data. Further, the
13 initial CBC value can be modified to incorporate the address of the data to force
14 the decrypted data to be placed at a specific location in order to decrypt correctly.

15 Alternatively, if the PLD allowed part of a design to be encrypted and part to
16 be unencrypted, the attacker could add an unencrypted portion to the encrypted
17 portion that would read out information about the encrypted portion of the
18 design. Thus, additional security is achieved by permitting the design to be totally
19 encrypted or totally unencrypted, but not to be mixed. Further to this, in one
20 embodiment, when data are being encrypted, additional security is provided by
21 allowing only a single full-chip configuration following a single starting address
22 for the configuration data.

23 Further, in order to allow convenient testing and debugging and to allow the
24 PLD manufacturer to communicate freely with its customers (the designers who
25 produce the designs for configuring the PLD), the PLD has both encrypted and
26 unencrypted modes of operating, and when operating in the encrypted mode,
27 parts of the configuration bitstream that control loading of the configuration data
28 into the PLD are still not encrypted.

29 As another mode of attack, if the PLD manufacturer gives information freely
30 about the configuration bitstream format, including header information and

addresses for loading configuration data, and gives information about the encryption method used, encrypting this well known information would expose the encryption key to possible discovery. Such exposure is avoided by encrypting only the actual configuration data and leaving control information unencrypted.

If the PLD manufacturer allows the key memory to be used in both secure and non-secure modes, an attacker could simply learn the keys by placing the key memory into non-secure mode and reading out the keys. To avoid such attack, the PLD manufacturer includes a circuit that causes all keys plus any configuration data loaded into the PLD to be erased when the key memory is moved to non-secure mode.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 shows functional relationships in a prior art FPGA.

Fig. 2a, 2b, 2c, and 2d show bitstream format and commands that can be included in a prior art bitstream.

Fig. 3 shows functional relationships in an FPGA according to one embodiment of the present invention.

Fig. 4a, 4b, 4c, and 4d show bitstream format and commands that can be included in a bitstream of the present invention.

Fig. 5a and 5b show example unencrypted and encrypted bitstreams.

Fig. 6 shows configuration logic 29 and the lines in bus 27 and bus 28 leading to decryptor 24.

Fig. 7a shows the modified starting value for outer cipher block chaining with triple encryption used in one embodiment of the invention.

Fig. 7b shows the corresponding starting value and decryption process used with Fig. 7a.

Fig. 8 shows flow of the operations for processing a bitstream.

Fig. 9 shows a state machine implemented by decryptor 24 to evaluate key order.

Fig. 10a shows the structure of key memory 23 of Fig. 3.

1 Fig. 10b shows the structure of the memory cells of Fig. 10a.

2 Fig. 11 shows the steps performed by control logic 23a of Fig. 10a to erase
3 keys when made non-secure.

4 Fig. 12 shows in more detail the battery supply switch of Fig. 10a.

5 Figs. 13 and 14 show the level shift circuit and voltage detection circuit of the
6 battery supply switch of Fig. 12.

7 Fig. 15 shows a state machine for erasing a design when a secure mode is
8 exited.

9 Fig. 16 shows a block diagram of elements for loading configuration memory
10 and reading back configuration, including lines disabled when encryption is
11 present.

12
13 DETAILED DESCRIPTION

14 Fig. 1 shows a prior art structure for an FPGA 10. The FPGA includes
15 programmable logic 11, typically comprising (1) logic blocks with lookup table
16 combinatorial logic function generators, flip flops for storing lookup table outputs
17 and other values, and multiplexers and logic gates for enhancing the logic ability
18 of the programmable logic (2) routing lines and programmable interconnection
19 points for routing signals around the FPGA, and (3) input/output blocks for
20 driving signals between the routing lines and the external pins of the FPGA.

21 The FPGA also includes configuration memory 12 for turning on routing
22 transistors, controlling multiplexers, storing lookup tables and controlling the
23 input/output blocks, all of this for the purpose of configuring the FPGA to
24 perform the function desired by the designer(s). Bus 16 connects configuration
25 memory 12 to programmable logic 11 and is typically a distributed set of control
26 lines located throughout the FPGA. Some Xilinx products (e.g. XC6200) have
27 included a bus 17 by which programmable logic 11 causes configuration logic 14
28 to send programming information to configuration memory 12. Such a structure
29 is described by Kean in U.S. Patent 5,705,938.

Fig. 2b shows the default format (format type 001) for a 32-bit header word in the bitstream used in the Virtex(R) devices available from Xilinx, Inc. (Virtex is a registered trademark of Xilinx, Inc., assignee of the present invention). This format includes three bits to indicate the format type (001), two bits to specify an op code, 16 bits for a configuration logic register address, and 11 bits for a word

count. The op code can designate a read operation, a write operation, or no operation. For example, 00 can designate no operation, 01 can designate read and 10 can designate write. The 11 bits for word count can specify 2^{11} words or 2048 words. As shown in Fig. 2c, if the word count is greater than this, the word count bits in format type 001 are set to 00000000000 and the header of format type 001 is followed by a header of format type 2. Format type 2 uses 27 bits to specify word count, and can thus specify 2^{27} words or 2.68 million words.

Fig. 2d shows the kinds of control information that can be loaded into the registers of Configuration Logic 14 by headers for a Virtex bitstream. For example, a header (of format 001) having the configuration logic register address 0000 specifies that the next 32-bit data word should be loaded into the cyclic redundancy check (CRC) register. (Virtex devices use a 16-bit cyclic redundancy check value so some bits will be padded with 0's.) If the header includes an address 0001, the next data will be loaded into the Frame Address register in order to specify a frame (column) in configuration memory 12 to receive or provide data.

The Configuration Logic Register address (16 bits) shown in Fig. 2b provides the 4-bit values shown in the left column of Fig. 2d that select one of the registers in configuration logic 14 (Fig. 1) into which to place the next 32-bit data word. The Frame Length register (address 1011) specifies the length of the frame into which the configuration data will be loaded. (Frame length, or column height, depends upon the size of the PLD. Larger PLDs usually have taller columns or longer frames. Specifying the frame length in the bitstream and storing the frame length in a register rather than providing a different structure in the PLD for placing the data words into frames allows the internal configuration logic to be identical for PLDs of different sizes.)

For readback, a read command is placed in the op code field and the Frame Data Output register is addressed, followed by a Word Count (using Command Header Format 2 if necessary). The specified number of words is read back from configuration memory 12, starting at the address specified in the Frame Address

1 register, and shifted out on either configuration access port 21 or JTAG port 20.
2 (Readback data is returned to the port that issued the readback instruction) .

3 Specifying a word count in a bitstream header or pair of headers (Figs. 2b
4 and 2c) sets a counter that counts down as the data words are loaded. For many
5 configuration logic register addresses the word count is 1. But if the bitstream
6 header has a configuration logic address of 0010 or 0011 to indicate configuration
7 data are being loaded in or read back, the word count will be much larger. This is
8 when header format 2 of Fig. 2c is used. Data loaded into configuration memory
9 12 through the frame data input register (address 0010) or read out through the
10 frame data output register (address 0011) is called the design data because it
11 causes the FPGA to implement a design or shows the status of a design. The other
12 register data are control data since they control how the configuration logic
13 behaves while the logic is being configured or read back.

14 Further detail about configuration of Virtex devices can be found in the
15 "Virtex Configuration Guide" published October 9, 2000 by Xilinx, Inc. (assignee
16 of the present invention), 2100 Logic Drive, San Jose, CA 95124. This
17 configuration guide is incorporated herein by reference.

18 Configuration logic 14 typically performs a cyclic redundancy check on a
19 configuration bitstream coming in (see Erickson, U.S. Patent 5,321,704
20 incorporated herein by reference, or see pages 39 through 40 of the above
21 referenced Virtex Configuration Guide), reads header bits indicating the frame
22 length of the part being configured and the word count of the configuration data,
23 reads address instructions identifying where to load configuration data, collects
24 frames of configuration data and loads them into columns of configuration
25 memory 12 indicated in the addresses. Configuration logic 14 also controls
26 readback of configuration data and flip flop values from configuration memory 12
27 to an external location. In a Virtex FPGA available from Xilinx, Inc., readback can
28 be done through either JTAG port 20 or through configuration access port 21.

29 Configuration logic 14 can also receive configuration data from
30 programmable logic 11. More information about prior art FPGA structures in

1 which part of the FPGA configures another part of the FPGA can be found in
2 Kean, U.S. Patent 5,705,938. More information about architectures of FPGAs
3 similar to the Virtex architecture can be found in Young et al., U.S. Patent
4 5,914,616. Both patents are incorporated herein by reference. The format of a
5 bitstream used with the Virtex product available from Xilinx, Inc., assignee of the
6 present invention, is described in an Application Note, XAPP138, entitled "Virtex
7 FPGA Series Configuration and Readback" available from Xilinx, Inc., 2100 Logic
8 Drive, San Jose, CA 95124 published Oct. 4, 2000.

9
10 PLD with Decryption

11 Fig. 3 shows a block diagram of an FPGA (a type of PLD) according to one
12 embodiment of the present invention. Some elements are the same as shown in
13 Fig. 1, are given the same reference numbers, and not explained again. In
14 addition, Fig. 3 includes an expanded configuration logic unit 29, a decryptor 24
15 and a key memory 23. Fig. 3 shows an embodiment in which key memory 23 is
16 loaded on bus 25 from JTAG access port 20. In other embodiments, key memory
17 23 is loaded through another port. Bus 25 carries data, addresses, and control
18 signals to perform write and read operations and allows programming of the
19 decryption keys from JTAG port 20. In one embodiment, bus 26 allows
20 programming of the keys from the configuration port. In another embodiment,
21 bus 26 is eliminated. In yet another embodiment, bus 26 is present and bus 25 is
22 eliminated. In an embodiment described further herein, bus 26 carries security
23 data from key memory 23 to configuration logic 29. In one embodiment, bus 27
24 carries encrypted configuration data from configuration logic 29 to decryptor 24
25 and carries decrypted configuration data back to configuration logic 29. Bus 28
26 allows decryptor 24 to access the keys for decrypting data. When the structure of
27 Fig. 3 is being loaded with encrypted data, an attacker who monitors the bitstream
28 as it is being loaded receives only the encrypted bitstream and can not learn the
29 user's design by this method.

1 Partially Encrypted Bitstream

2 According to another aspect of the invention, the bitstream comprises two
3 portions, a data portion representing the user's design that can be encrypted or
4 not, and a control portion controlling loading of the bitstream (for example giving
5 addresses of columns in the PLD into which successive portions of the bitstream
6 are to be loaded, providing a cyclic redundancy check (CRC) code for checking
7 reliability of the loading operation, and a starter number for cipher block chaining
8 (CBC), a technique that prevents a "dictionary attack" where the decrypted data
9 can be deduced from the frequency of occurrence of the encrypted data). In a
10 preferred embodiment of the invention, the data portion may be encrypted but the
11 control portion is unencrypted. This provides additional security because the PLD
12 manufacturer needs to describe freely the control features of the bitstream, and if
13 this relatively well known control information were encrypted, an attacker might
14 be able to decrypt this information and use this information to decrypt the entire
15 bitstream. Further, keeping the control portion of the bitstream unencrypted
16 makes it easier for the PLD to use the information.

17 In another embodiment, used when the order of addresses in which
18 configuration data is loaded may be useful to an attacker in analyzing the design,
19 the address of the configuration data is also encrypted, but other control
20 information in the configuration bitstream remains unencrypted.

21
22 Bitstream Format

23 Figs. 4a-4d illustrate differences in bitstream format and registers of
24 configuration logic 29 in comparison to the format and registers of configuration
25 logic 14 of the prior art product shown in Figs. 2a-2d. As shown in Fig. 4a, the
26 bitstream still includes header words followed by data words. In a typical
27 configuration, several control data words will be loaded into registers before
28 encrypted configuration data begins. Fig. 4a shows an example in which three
29 header words Header A, Header B, and Header C are each followed by three
30 unencrypted control data words Data A, Data B, and Data C. (In an actual

As shown in Figs. 4b and 4c, a fourth op code has been added. In addition to the values 00 for no operation, 01 and 10 for read and write without decryption, the new value 11 specifies that writing is to be with decryption. (It is not important what code or what method is used to specify that decryption is to be used or even that it is specified through an op code. It is just important that optional encryption and decryption be allowed and indicated, so that designers can make use of this option. In the embodiment of Fig. 4d, two new configuration logic registers are added. Shown at addresses 1100 and 1101 are the register for holding a cipher block chaining (CBC) starter value and the address for the initial encryption key.

According to another aspect of the invention, a PLD can accept both encrypted and unencrypted data portions of the bitstream. The control portion of the bitstream indicates whether the data portion of the bitstream is encrypted. If the data portion of the bitstream is encrypted, it is diverted within the PLD to a decryptor and after decryption is used to configure the PLD. If unencrypted, it is not diverted, and is used directly to configure the PLD.

14

1 times, it may be necessary to make all portions of the bitstream visible, and to
2 allow the PLD to be partly reconfigured.

3 Figs. 5a and 5b show example bitstream portions representing the same
4 design, first unencrypted and then encrypted, to illustrate the differences between
5 an unencrypted bitstream and an encrypted bitstream in one embodiment of the
6 invention. An actual bitstream includes the 0's and 1's at the right of the figures
7 and none of the text at the left. The text at the left is provided to explain the
8 meaning of the bits to the right. These bitstream portions use the commands
9 illustrated in Figs. 4b-4d. In order to emphasize the differences between the
10 unencrypted version of Fig. 5a and the encrypted version of Fig. 5b, the
11 differences are shown in bold.

12 Looking at Fig. 5a, after a dummy word (a constant high signal interpreted
13 as all 1's) and a sync word with a specified pattern of 1's and 0's, the next word is
14 of type 001 with an op code of 10, has an address of 0000000000010000 and a word
15 count of 00000000001. Thus this word addresses the command register CMD and
16 specifies that one word will be written there. Fig. 5a has been annotated to the left
17 of the bitstream to indicate that this word is Type 1 and indicates to write 1 word
18 to CMD. The following word 111 is the data to be placed in command register
19 CMD, and resets a CRC (cyclic redundancy check) register. (In a preferred
20 embodiment, the PLD includes a circuit, not shown, such as described by
21 Erickson in U.S. Patent 5,598,424 to calculate a CRC value from the bitstream as
22 the bitstream is being loaded, and protects against glitches in the bitstream
23 voltages that might cause incorrect bits to be loaded.) Next, a header word
24 specifies that the format is again type 1 and it specifies to write 1 word to the
25 frame length register FLR. The data word that follows, 11001, specifies the frame
26 length (25 words). Similarly, several additional header and data words follow,
27 including the header specifying the word to be written to the frame address
28 register FAR. In this case, the following data word indicates data will start at
29 address 0. Finally, after these registers have been loaded, a command comes to
30 write data to the frame data input register FDRI, and since quite a bit of data will

09724975-112800

1 be written, the word count is given as 00000000000 and a header of type 2 specifies
2 that 10530 words will be written to the FDRI register. This is the actual design
3 data that causes the PLD to be configured. Thus the next 10530 words in the
4 bitstream are design data. Finally, to assure that data have been loaded correctly,
5 the CRC value calculated by the device that originated the configuration data is
6 loaded and compared to the CRC value that has been calculated by the PLD.
7 Additional commands and data are loaded in order to indicate that configuration
8 is complete and to move the PLD into operation mode.

9 Fig. 5b is similar to Fig. 5a, and differs only where the data and annotations
10 are shown in bold. In Fig. 5b, the data are encrypted, and additional commands
11 are used to provide the initial key address and to write two words (64 bits) to the
12 CBC (cipher block chaining) register. Next, a type 1 header includes the op code
13 11 and indicates that data will be decrypted before being written to frame data
14 input register FDRI. A type 2 header follows, again with the op code 11, giving
15 the instruction that 10530 words are to be decrypted and written to data input
16 register FDRI. The 10530 encrypted data words then follow. Then the CRC word
17 follows for confirming that the (encrypted) data were loaded correctly. Finally,
18 the additional commands and data are sent, and place the PLD into operation
19 mode if all is correct.

21 Decryption Process

22 Fig. 6 shows how optional decryption is accomplished in one embodiment.
23 Fig. 6 shows the detail of configuration logic 29 and of buses 27 and 28 leading
24 into decryptor 24. Bus 27 includes the following:

- 25 •the 3-bit initial decryption key address "Init_key_addr" taken from
26 register address 1101 (Fig. 4d) in configuration logic 29,
- 27 •the 64-bit modified cipher block chaining value "modCBC". This value is
28 formed by replacing the lower order bits of the 64-bit CBC value
29 taken from register address 1100 (Fig. 4d) in configuration logic 29
30 with the 22-bit Frame Address value specified in Register 0001.

- 1 •the 64 lines "Encrypted_data" for loading encrypted data, taken from the
- 2 bitstream,
- 3 •the 64 lines "Decrypted_data" for returning the decrypted data produced
- 4 by decryptor 24 to configuration logic 29,
- 5 •a line for the signal "Enc_data_rdy" that tells decryptor 24 that data is on
- 6 the "Encrypted_data_lines and that decryptor 24 can start
- 7 decrypting,
- 8 •a line for the signal "Dec_data_rdy" that tells configuration logic 29 that
- 9 decryption on a 64-bit word is complete and is available on the
- 10 "Decrypted_data" lines, and
- 11 •a Bad_key_set line used by decryptor 24 to cause configuration logic 29 to
- 12 abort the configuration and set a status register accordingly when
- 13 the keys have not been used as specified, for example, by the bits in
- 14 key memory that designate whether the keys are to be first, middle,
- 15 or last of a set. In the embodiment shown in Fig. 4d, the status
- 16 register is at address 0111, and the Bad_key_set error is indicated by
- 17 storing a logic 1 in one of the bits.
- 18 Bus 28 is comprised of the following:
- 19 •3 lines for the key address, which is initially the key address provided
- 20 from bus 27, but which is updated each time a new key is used,
- 21 •56 lines for the decryption key, and
- 22 •2 lines for indicating whether the decryption key is the first, middle, last,
- 23 or only key to be used.

24 Preventing Design Relocation

25 One potential attack on a design in an encrypted bitstream is to change the
26 frame address register (starting address) in the encrypted bitstream so that when
27 it is decrypted it is loaded into a portion of the FPGA visible when the FPGA is
28 being used. In some designs the content of the block RAM is visible. In all
29 designs the configuration of the input/output ports is visible and therefore the
30

THE UNIVERSITY OF CHICAGO

As shown in Fig. 7b, the PLD must perform the reverse process to derive the decrypted words. For the above encryption sequence, the decryption sequence would be decryption step dec_1 using key 3, then encryption step enc_2 using key 2, then decryption step dec_3 using key 1. Importantly, part of the initial value for generating Decrypted Word1 is to use the same frame address for both encryption

1 and decryption. The PLD, not the bitstream, generates the modified CBC value
2 from the frame address stored in the frame address register, which is also used to
3 specify the frame of configuration memory 12 into which configuration data are to
4 be loaded. So if an attacker changes the frame address into which the data are to
5 be loaded, the modified CBC value changes accordingly, and the configuration
6 data are not correctly decrypted.

7 The XOR step produces the original data that was in the designer's bitstream
8 before it was encrypted. Original Word1 = Decrypted Word1, for example. This
9 decrypted configuration data is sent on bus 27 (Fig. 3) to configuration logic 29.

11 Configuration Logic 29

Configuration logic 29 includes the structures to support optional encryption as well as the structures to prevent design relocation and a single key attack. As shown in Fig. 6, configuration logic 29 includes a holding register 292, control logic 291, configuration registers (FDRI, FAR, CRC, and init CBC are shown), decryptor 24 interface multiplexers 294 and 295, 64-bit assembly register 297, and registers 298 and 299 (for interfacing with configuration access port 21). A 64-bit shift register 299 receives data from configuration access port 21, which can be a single pin for 1-bit wide data or 8 pins for 8-bit wide data. This data is loaded into 64-bit shift register 299 until register 299 is full. Then these 64 bits are preferably shifted in parallel into 64-bit transfer register 298. From there, multiplexer 296b alternately selects right and left 32-bit words, and multiplexer 296a moves the data 32 bits at a time either into holding register 292 or alternately into High and Low portions of assembly register 297 as controlled by control line M. When loading of the bitstream begins, line M and a clock signal not shown cause multiplexers 296a and 296b to move data from 64-bit transfer register 298 to holding register 292. From there these words are applied to control logic 291. If the word is a header, control logic 291 interprets the word. If the op code indicates the data to follow are to be written unencrypted, control logic 291 places an address on bus G to select a register, places a signal on line L to cause multiplexer 294 to connect bus B

1 to bus D, and applies the following word on bus B. On the next clock signal (clock
2 signals are not shown), the data on bus D are loaded into the addressed register.
3 All registers shown in Fig. 4d can be loaded this way. The init CBC register for
4 loading the initial cipher block chaining value is a 64-bit register and receives two
5 consecutive 32-bit words, as shown in Fig. 5b and discussed above.

6 A modified CBC value formed from (1) the original CBC value stored in the
7 init CBC register and (2) the initial frame address stored in the FAR register is
8 available to decryptor 24. In one embodiment, the initial frame address in the
9 FAR register uses no more than 32 bits while the init CBC value uses 64 bits. In
10 the embodiment of Fig. 6, the 64-bit bus providing the modified CBC value
11 includes 22 bits from the frame address register FAR and 42 bits from the init CBC
12 register. Important to the security provided by the present invention, note that
13 this value depends upon where configuration data will be loaded. If an attacker
14 were to try to load encrypted data into a different place by changing the contents
15 of the FAR register, the modCBC value fed to decryptor 24 would also change.

16 When the op code command to decrypt a number of words of configuration
17 data is received by control logic 291, the decryption process begins. Control line
18 M causes multiplexer 296a to apply data from transfer register 298 to bus A
19 leading to assembly register 297. Control bus H alternately connects bus A to the
20 High[31:0] and Low[31:0] portions of encrypted data register 297 to form a 64-bit
21 word to be decrypted. Control logic 291 then asserts the Enc_data_rdy signal,
22 which causes decryptor 24 to decrypt the data in register 297.

23 To perform the decryption, decryptor 24 applies a key address KeyAddr on
24 bus 28 to key memory 23 (Fig. 3). This causes key memory 23 to return the 56-bit
25 key in that address on the 56-bit Key lines. It also causes key memory 23 to return
26 two additional bits "Order" also stored in the key data at that address. For the
27 first decryption key, these two bits must indicate that this is a first key or an only
28 key. If not, decryptor 24 asserts the Bad_key_set signal, which causes control logic
29 29 to abort the configuration operation. If these two bits indicate the key is a first
30 or only key, decryptor 24 performs the decryption, using for example the well

1 known DES algorithm (described by Schneier, *ibid*). If the key isn't an only key,
2 decryptor 24 then gets the key at the next address in key memory 23, and checks
3 to see if the two Order bits indicate it is a middle or last key. If not, the
4 Bad_key_set signal is asserted and the configuration is aborted. If so, decryption
5 is performed. If it is a middle key, another round of decryption is done. If it is the
6 last key, decryptor 24 forms the XOR function of the decrypted word and the
7 value modCBC. Decryptor 24 then places the resultant value on the 64-bit
8 Decrypted_data bus and asserts the Dec_data_rdy signal. This causes control logic
9 291 to place signals on control line K to cause multiplexer 295 to break the 64-bit
10 word into two sequential 32-bit words. Control logic 291 places a signal on line L
11 to cause multiplexer 294 to forward the 32-bit words of decrypted data to bus D.
12 Control logic 291 also places address signals on bus G to address frame data input
13 register FDRI. The next clock signal moves the decrypted data to bus E where it is
14 loaded into the frame register and when the frame register is full, eventually
15 shifted into configuration memory 12 at the address indicated in the FAR register.

16 The modCBC value is used only once in the decryption operation.
17 Subsequent 64-bit words of encrypted data are decrypted and then chained using
18 the previously decrypted data for the XOR operation. (The value stored in the
19 FAR register is also used only once to select a frame address. Subsequently, the
20 frame address is simply incremented every time a frame is filled.)

21

22 Flow of Operations

23 Fig. 8 indicates the flow of operations performed by configuration logic 29
24 and decryptor 24. Configuration logic 29 begins at step 70 by loading the bitstream
25 headers and placing the corresponding data into configuration logic registers
26 shown in Fig. 4b, including determining bitstream length. At step 71, as a further
27 part of the start-up sequence, configuration logic 29 reads the first configuration
28 memory address. Recall that the bitstream format includes an op code that
29 indicates whether encryption is being used. Step 72 branches on the op code
30 value. If encryption is not used, the process is shown on the left portion of Fig. 8.

At step 84, decryptor 24 decrypts the 64-bit data with the 56-bit key using, for example, the DES algorithm. The DES algorithm is described in the above-mentioned book by Bruce Schneier at pages 265 to 278. Other encryption algorithms may also be used, for example, the advanced encryption standard AES.

3 Step 85 determines whether more keys are to be used. The two bits that
4 indicate whether the key is first, middle, last, or only key are examined to
5 determine whether this is the last key, and if not, the key address is incremented
6 and decryptor 24 addresses the next key in memory 23.

At step 88, this decrypted configuration data is sent on bus 27 (Fig. 3) to configuration logic 29. Configuration logic 29 calculates an updated cyclic redundancy check value to be compared with the cyclic redundancy value stored in the CRC register at the end of the loading process. If configuration logic 29 has been set to use encryption, a multiplexer in configuration logic 29 forwards this decrypted configuration data to the addressed column of configuration memory 12.

28 Finally, when step 89 indicates the counter is finished, at step 90, a CRC
29 (cyclic redundancy check) value in the bitstream is compared with a CRC value
30 calculated as the bitstream is loaded. If the values agree, configuration is complete

092463-166000

1 and the FPGA goes into operation. If the values do not agree, a loading error has
2 occurred and the entire configuration process is aborted.

3 4 Evaluating Key Order - Preventing Single Key Attack

5 Fig. 9 shows a state machine implemented by decryptor 24 to evaluate key
6 order. The state machine remains in state S1 until the Enc_data_ready signal is
7 activated. This signal indicates decryption can begin and moves to decision state
8 Q1 where decryptor 24 applies the address specified by the address Init_key_addr
9 on bus 27 to bus 28, reads back a key and a key order, and from the two bits of key
10 order data determines whether the key is a first or only key. If not, decryptor 24
11 sends the Bad_key_set signal to control logic 291 and causes configuration logic 29
12 to abort the configuration. If the address is first or only, decryptor 24 goes to state
13 S3, which decrypts the data. Then the state machine goes to decision state Q2,
14 which determines whether the key is last or only. If so, decryption is complete
15 and at state S4 decryptor 24 returns the decrypted data to configuration logic 29.
16 If not, in state S5, decryptor 24 increments the key address, and gets the new key.
17 The state machine asks question Q3 to determine whether the next key is a middle
18 or last key. If not, state S2 causes the configuration to abort. If the key is middle
19 or last, the state machine returns to state S3 to decrypt the data again. In another
20 embodiment, in state S4 decryptor 24 also performs the step of XORing the
21 decrypted data with a CBC value.

22 The benefit of storing the key order within the keys is that an attacker can
23 not implement a single key attack because the attacker can not prevent decryptor
24 24 from using all the keys specified by key memory 23 (as intended by the
25 designer) when performing decryption. It is not necessary to ask the second and
26 third questions Q2 and Q3 to protect against an attacker using a single key attack,
27 since the key order is stored within the key data inside the PLD. However, it is
28 beneficial to the designer or board tester who loads the keys to ask all three
29 questions to make sure that each key has been labeled correctly when it is loaded.

1 In one embodiment, decryptor 24 uses the triple DES standard with a
2 decryption-encryption-decryption sequence, alternating the algorithm (only
3 slightly) each time another key is used. Such a combination is in accordance with
4 the ANSI X9.52 1998 Triple DES standard. In another embodiment, decryption is
5 used each time.

6
7 Key Memory 23

8 The circuit shown in Fig. 10a includes three components: battery supply
9 switch 22, control logic 23a and key registers 23b. Control logic circuit 23a and
10 key registers 23b comprise key memory 23 of Fig. 3. In the embodiment of Fig.
11 10a, key registers 23b comprise six 64-bit words. Of course, other key memory
12 sizes may alternatively be used. In other embodiments, there may be far more
13 than six keys stored in key memory 23, and more than 3 bits needed to give the
14 address of the key to be used. The power supply for key registers 23b comes from
15 battery supply switch 22 on line VSWITCH. When key memory supply voltage
16 VCCI is insufficient or not present, battery supply switch 22 applies the battery
17 backup voltage VBATT to the VSWITCH line so that VSWITCH carries a positive
18 voltage.

19 In this embodiment each key register has 64 memory cells. Each cell receives
20 a write enable signal WE, that when high causes data to be written to the cell and
21 when low causes data in the cell to be held. Cells in one register have a common
22 write enable signal WE. When the PLD supply voltage (different from VCCI) is
23 absent such that the WE signals are not actively driven, weak pull-down
24 transistors such as T1 pull down the WE signal so that none of the key memory
25 registers can be addressed, and none of the memory cells are disturbed.

26 In one embodiment, the JTAG port of a PLD is used to load decryption keys
27 into the PLD. The memory cell supply voltage is at the device voltage level of
28 VCCI during normal operation, and in one embodiment this level is between 3.0
29 and 3.6 volts. Signals applied to the JTAG port may be several different voltages.
30 Also, there may be several different internal voltages. Thus voltage translation is

Control logic circuit 23a receives signals from JTAG bus 25 (also shown in Fig. 3). JTAG bus 25 includes control signals for writing, reading, setting the secure mode, and data and address buses. This interface conforms to the IEEE 1532 JTAG standard. Before key memory 23 can be accessed through JTAG bus 25, the security status (bus 26) is placed in non-secure mode, which can be done using the ISC_PROGRAM_SECURITY instruction (see Fig. 10a) and applying logic 1 to bit 0 of the key data bus. Key memory 23 is written to and read (for verification) from JTAG bus 25 using the ISC_PROGRAM and ISC_READ instructions of the IEEE 1532 standard. Control logic 23a includes a decoder for decoding the 3-bit address signal ADDR from JTAG bus 25 to produce a low-going pulse on the addressed one of write strobe lines ws_b[5:0] if the ISC_PROGRAM instruction appears on JTAG bus 25, or a high signal on the addressed one of read select lines rsel[5:0] if the ISC_READ instruction appears on JTAG bus 25. One of the six 64-bit words can be read by applying a high signal to one of the six read select lines rsel[5:0], which causes read multiplexer 23d to place the selected word on the 64 output lines q[63:0]. Only one of the write select lines or read select lines is selected at one time. When no read select signal is asserted, a high park_low signal causes 64 transistors 23e to pull down the 64 lines q[63:0] and prevent these lines from floating.

26

To communicate whether key memory 23 is in secure mode, control logic 23a sends a secure mode signal on bus 26 (may be a single line) to configuration logic 29 to indicate that key memory 23 is operating in secure mode. If this signal switches to non-secure mode, configuration logic 29 clears the design from configuration memory 12. Note that an unencrypted bitstream may be loaded by

3

For decryption, these keys are used in succession (reverse order) to repeatedly decrypt the encrypted bitstream. Further to this, if more keys are stored in the PLD than are used for decrypting a particular design, the encrypted bitstream may include in an unencrypted portion an indication of how many keys

1 are to be used, and the address of the first key. Such an embodiment may make it
2 easier for an attacker to decrypt the bitstream because the attacker need only deal
3 with one key at a time. Alternatively, the keys themselves may indicate whether
4 they are the first, middle, last, or only keys. Thus the same PLD can at different
5 times be programmed to perform different functions (configured with different
6 designs), and information about the values of the different keys can be made
7 available to only one or some of the designers. Thus a first designer may not learn
8 about a second design even though both designs are implemented in the same
9 PLD (at different times).

10 Regarding Fig. 3, configuration logic 29 includes additional logic beyond
11 configuration logic 14 of Fig. 1. As in the structure of Fig. 1, the bitstream on
12 configuration access port 21 is treated as words, in one embodiment 32-bit words.
13 Several of the words, usually at or near the beginning of the bitstream, contain
14 header information, for example length of the bitstream, starting address for the
15 configuration data. New to the bitstream of the present invention is an indication
16 as to whether the bitstream is encrypted, and the address of a key for decrypting
17 configuration data in the bitstream.

18 19 Battery Backed up Memory

20 Values stored in key memory 23 are preferably retained by a battery when
21 power to the FPGA is removed.

22 Further, other memories than encryption keys can also be backed up using a
23 battery supply switch such as switch 22. In particular, a PLD can be
24 manufactured in which the VSWITCH voltage supply is routed to all flip flops in
25 the PLD if the purpose is to preserve data generated by the PLD when the PLD is
26 powered down. And if the purpose is to also preserve configuration of the PLD
27 when the PLD is powered down, configuration memory 12 (Fig. 3) may
28 alternatively be powered from VSWITCH, though such an embodiment requires
29 considerably more battery power than does powering just the flip flops in the

31

Key Not Available to Purchaser of a Product Containing the Configured PLD

In order to prevent an attacker from learning the design that has been used to configure the PLD, several additional steps may be taken.

According to another aspect, a key is loaded into the PLD before sale of a system incorporating the PLD, such that after sale of a system including the PLD, the design can be loaded into the PLD and used, but an attacker can not learn the value stored in the key (or keys). Thus the unencrypted design can not be read or copied. To achieve this security, several steps are taken.

Secure Mode Preservation (Tamper-proofing)

In one embodiment, there are two security flags in configuration logic 29 of the PLD. One indicates whether the decryption keys are secured, and the other indicates whether the design is a decrypted design and must be protected. If JTAG logic 13 (Fig. 3) selects secure mode with the ISC_PROGRAM_SECURITY instruction, a secure_key flag in control logic 23a (Fig. 10a) is set. If the bitstream loaded into the PLD has the indication that design data in the bitstream is encrypted, a secure_design flag in configuration logic 29 (not shown) is set. If either flag is later unset, the entire configuration memory is cleared, thereby removing the decrypted design. If the secure_key flag is reset (by an ISC_PROGRAM_SECURITY instruction), then the keys are also erased.

Fig. 15 shows a state machine for performing the design clearing function. When the secure_design flag is set, the state machine enters state S1. This state monitors a change from secure to non-secure mode of the secure_design flag. As long as the secure-design mode continues, the state machine stays in state S1. Once a change occurs, the state machine enters state S2 and the data shift registers for shifting data into configuration memory 12 are reset, thereby placing zeroes on all data lines for the configuration memory bits. Next, the state machine moves to state S3 where the word line of the addressed frame is asserted. This results in the zeros on the data shift register lines being written into the memory bits at the addressed frame. If question Q1 indicates there are more frames to be addressed,

1 the state machine moves to state S4 where the frame address is advanced and the
2 state machine returns to state S3. When question Q1 indicates there are no more
3 frames to be addressed, the process is done and the configuration memory is
4 cleared.

It is also necessary to protect the keys from being accessed by an attacker. Loading of the keys is performed before a system containing the design is made available to an end customer. When designers are in the process of developing the design, they may wish to operate the PLD in a non-secure mode for debugging. In order to allow for this debugging operation and also to preserve security of the keys, the key loading process begins in a non-secure mode by clearing all key registers. A secure key flag must be kept in the non-secure mode while keys are loaded and while the keys are read back for verification. The secure key flag may also be kept in the non-secure mode while a configuration bitstream is loaded and decrypted. But once the secure key flag is set, returning the secure key flag to the non-secure mode clears all keys and also initiates operation of the state machine of Fig. 15. So, not only are the keys cleared, but the configuration is also cleared.

18 Readback Attack and Readback Disabled

19 Some FPGAs allow a bitstream to be read back out of the FPGA so that a
20 user may debug a design or may obtain state machine information from flip flops
21 in the FPGA. Unless the design were re-encrypted for the read-back operation, the
22 act of reading back the bitstream would expose the unencrypted bitstream to
23 view.

Further security of the design is provided by disabling readback when an encrypted design is loaded into the FPGA. In one embodiment, readback is disabled only if the decryption keys are also secured.

Fig. 16 shows the block diagram of a structure for loading and reading back configuration memory. In one embodiment, configuration logic prevents readback when two conditions are present: (1) the security status line on data bus 26 (see Figs. 3 and 10) indicates that the keys are in a secure mode, and (2)

1 configuration logic 29 has responded to op codes in a configuration bitstream that
2 indicate the bitstream is encrypted. So if either the keys are not secured or the
3 bitstream is not encrypted, readback can be enabled. In other embodiments,
4 different conditions control whether readback can be enabled.

5 When configuration logic 29 receives in the bitstream a header indicating
6 that readback is to be performed, it sends on line 107 the frame address stored in
7 its frame address register, which is decoded by address decoder 110 to select the
8 addressed line of bus 109. Next, word line enable signal on line 108 is asserted,
9 which asserts the selected word line of bus 109 to allow memory cells addressed
10 by the selected word line to place their values on the n data lines 102 (n is the
11 frame length and is stored in configuration logic 29). Configuration logic 29 then
12 asserts the Load signal on line 104 to load the frame of data (in parallel) into data
13 shift register 101. Next, configuration logic 29 asserts the shift signal on line 105 to
14 cause data shift register 101 to shift out the frame of data in 32-bit words on bus
15 103 to the frame data output register (see Fig. 4d) and from there to an outgoing
16 bitstream on configuration access port 21 (Fig. 3).

17 If decryption is indicated in the bitstream, configuration logic 29 sets internal
18 flags to indicate this. If these flags are set and key memory 23 is in secure mode as
19 indicated by the security status signal on bus 26, then configuration logic 29
20 responds to a readback command in the bitstream by keeping the word line
21 enable signal on line 108 inactive and by keeping the load and shift signals on lines
22 104 and 105 inactive to prevent readback. However, if key memory 23 is not in
23 secure mode, even though the design may be encrypted, readback is allowed so
24 that testing and debugging are possible.

25 26 Partial Reconfiguration Attack and Prevention

27 Some FPGAs allow partial reconfiguration of the FPGA or allow different
28 parts of a design to be loaded into different parts of the FPGA using separate
29 starting addresses and separate write instructions. An attacker might attempt to
30 learn the design by partially reconfiguring the design to read contents of a block

1 RAM or flip flops directly to output ports or by adding a section to an existing
2 design to read out information that can be used to learn the design. For example,
3 the attacker might partially reconfigure the PLD with an unencrypted design
4 whose only purpose is to extract information about the encrypted design. Such a
5 Trojan Horse design could be loaded into the PLD with another bitstream or
6 attached to an existing encrypted bitstream. If the attacker was interested in
7 learning a state machine design loaded into block RAM of an FPGA, for example,
8 the Trojan Horse design could include logic to cycle through the addresses of the
9 block RAM and send the block RAM data contents to package pins.

10 In order to prevent an attacker from making such changes, if the original
11 design is encrypted, configuration logic 29 disallows partial reconfiguration once
12 configuration with decryption is started. Configuration logic 29 disallows a
13 further write instruction once a header with the decryption op code has been
14 processed. Also, configuration logic 29 disallows configuration with decryption
15 once configuration without encryption has been done. Configuration logic 29
16 accomplishes these restrictions by ignoring headers that write to configuration
17 memory after a decrypt instruction has been received and ignoring headers that
18 have a decrypt command if an unencrypted portion of a design has been loaded.
19 Thus, if any op code indicates that writing with decryption is being used, the PLD
20 will accept only a single write instruction.

21

22 Additional Embodiments

23 The above description of the drawings gives detail on a few embodiments.
24 However, many additional embodiments are also possible. For example, instead
25 of the cipher block chaining algorithm discussed above, one can use an encryption
26 method called cipher feedback mode in which data can be encrypted in units
27 smaller than the block size, for example one 8-bit byte at a time. This cipher-
28 feedback mode is described by Schneier, *ibid*, at pages 200-203.

29 In yet another embodiment, if encryption is used, all bitstreams must be
30 loaded starting at address 0. One implementation of this embodiment replaces

1 any address loaded into the starting frame address register FAR (Fig. 6) with
2 address 0 when an op code specifying encryption is received.

3 In still another embodiment, the starting address and the design data are
4 both encrypted. In this embodiment, it is possible to load several segments of
5 encrypted design data starting at different frame addresses, just as is possible with
6 unencrypted design data.

7 In another embodiment, the key data stored in a key memory such as key
8 memory 23 specifies the number of keys that will follow. In a variation on this
9 embodiment, the key data also specify the number of keys that precede the key. If
10 an attacker gives a key address other than the first key address intended by the
11 designer, the configuration may be aborted. Additionally, encryption will proceed
12 until the number of keys specified within the keys have been used.

13 In another embodiment, instead of allowing keys to be read back when the
14 key memory is in a non-secure mode, the keys include parity bits or CRC check
15 bits, and only these bits can be read back for verification that the key or keys were
16 loaded correctly. This embodiment allows keys known to one designer to be kept
17 secret from another designer, and is useful when the PLD is to be used at different
18 times for loading different designs.

19 Regarding the CRC checksum calculation discussed above, embodiments can
20 be provided in which the CRC checksum is calculated either before or after a
21 design is encrypted. Of course, if the checksum added to the bitstream is
22 calculated before the design data is encrypted, then a corresponding checksum
23 must be calculated within the PLD on the design data after it has been decrypted.
24 Likewise, if the checksum added to the bitstream is calculated after the design
25 data has been encrypted, then the PLD must calculate the corresponding
26 checksum on the received bitstream before the design data have been decrypted.

27 A further note regarding the process of loading the decryption keys, when
28 the process illustrated in Fig. 8 is used, it is not necessary to use a device
29 programmer for loading decryption keys. The keys may simply be loaded as part
30 of the board test procedure.

0082495-11800

13 Variations that have become obvious from the above description are
14 intended to be included in the scope of the invention.

13 Variations that have become obvious from the above description are
14 intended to be included in the scope of the invention.